

Personnel

Lecture 1

Introduction to Operating Systems

In this introduction, we will see why operating system is an indispensable piece of software that should be carefully studied.

Text: Operating Systems Concepts [OSC], 6th Ed. Silberschatz and Galvin. John Wiley and Sons Inc.

References:

- **Linux Device Drivers [LDD]**. Alessandro Bubini & Jonathan Corbet. O Reilly.
- **Understanding the Linux Kernel [ULK]**. Bovet & Cesati. O Reilly.

POS(0230A)

Students: CSIS and BBA(IS)/BEng(SE) double degree

Lecturer:

- Dr. Isaac To Kar Keung (kkto@csis.hku.hk), CYC407(2859-2173).

Tutors:

- Felix Cheung Kai Man (kmcheung@csis.hku.hk).
Consultation hour: Tue 1600–1700.
- Ken Tsang Cheuk Kan (cktsang@csis.hku.hk).
Consultation hour: Tue 1700–1800.
- Zhang Ming Hua (mhzhang@csis.hku.hk).
Consultation hour: Tue 1500–1600.

Web site: <http://www.csis.hku.hk/~c0230a/>

POS(0230A)-1.1

Calling for help

If you have difficulties, you can do one of the followings:
In the order of preference...

- Post a news article to our **newsgroup**, hku.csis.CSIS0230A. This is preferred: all of you can answer, and all of you benefit from the answer. Questions asked in any other means may be answered (only) in newsgroup, so **please keep watching the newsgroup**.
- Send us a **mail** at c0230a@csis.hku.hk. The mail will end up in the mailbox of me and all tutors.
- Contact us during **tutorials**.
- Contact us directly. Send a personal mail before visiting our office to make an **appointment**.
This is the order in which I deal with them. I.e., news are first cleared, and only when all news are dealt with I'll deal with mails to c0230a, and only after that I'll look at personal mails.

POS(0230A)-1.2

Mode of instruction

Lectures

- 2 hours/week (Every Monday, 1600–1800). Voluntary.
- Primarily uni-directional flow of information. I teach, you listen.
But don't hesitate to ask questions.

Small Group Tutorials

- Provides supervised lab experience, 1 hour/week. **Compulsory**.
Expect overrun and spend 2 hours instead.
- **Study the reading material well** before going to tutorials.
- Indicate preference to the time for tutorial slot after this lecture.
Tutorials can't be large. So expect to make compromises: you might get a slot on an otherwise off day.

POS(0230A)-1.3

Lecture conduct

Lectures are free-form, relaxed but serious.

- **Talking** is allowed **only on course matters**.
- **Attend lectures** only if you are willing to learn. Remember that lectures are voluntary.
- **Everything** in the lectures **needs to be understood**, preferably during the lecture, and absolutely no later than the end of the day.
- **Ask** whenever you find something you don't understand. Don't delay until the end of the lecture, or even worse until an assignment comes.
- **Ask me to stop or repeat** if you need time to digest.

POS(0230A)-1.4

Assessment

Assignments (30%)

- 4 programming assignments, in Groups of 1–2.
- Don't ask us to debug your programs, but you can ask any question.
Well... debug at most once per group per assignment.

Quiz (10%). To be announced

- 30 min quiz of very easy questions. No problems if you have done the assignments and understands the tutorials well.
This is perhaps a big if, though.

Tutorial assessment (10%) during 13 weeks

- Subjective assessment by TAs and lecturer.
- No more than one absense. Each further absense results in a loss of $1/(\text{Number of tutorials} - 1)$ of all 50% coursework marks.

Exam (50%)

POS(0230A)-1.5

Policy: late submission, plagiarism

Late submission

- Requests for postponing deadlines will be considered only if it is made **at least 3 days** before deadline.
- Assignments late for at most **48 hours** will **score 80%** of the score it would score if it was submitted on-time.
- Assignments late for more than 48 hours are marked as normal, but will not contribute to the final score.

Plagiarism

- Any discussion within group are allowed.
- Discussions across group or to outside groups are limited to sharing the concepts. No actual code or numbers may be exchanged.
- Plagiarism is **punished as described by departmental guidelines**.

POS(0230A)-1.6

Exam/Study tips

- **No need to memorize anything** (exam is open book).
For those things you really need to remember, you'll find that you naturally remember them because you do it again and again.
- **Understand the concepts deeply**. Exam will be on concepts.
Try to imagine alternative ways to do things and figure out whether it is a good or bad alternative in terms of functionality and resources usage.
- **Keep up with lectures**.
- **Do the assignments early**. They greatly enhance the understanding of the course materials.
- **Read the book**. Exam will be on things that you have never met before, so the more experience you have, the better.
And don't expect you can read it during the exam.

POS(0230A)-1.7

Objectives

For students to learn:

- **What** is an operating system? What **services** it provides?
- **Why** the operating system is **designed** in those ways?
- **How** to **access** the services in a practical OS, namely Linux? What are the performance implications?
Other OS are similar, and it is better to learn deeply into one rather than learn superficial things in many. We just learn one which is most available.
- **How** these services are **implemented**?

POS(0230A)-1.8

The environment of our program

- Solving problems with C++: **compile** the program to **machine code**, and load and execute it by running a **command**.
- The computer then **loads** the program from the disk.
- The **CPU** (or "**processor**") of the computer then runs the machine code, one instruction at a time, to do the things we write.
- **Memory** is allocated for program **variables**, so that values are stored and retrieved.
- The program can access **files** and **devices** to get data in from and results out of the computer.

In short, **the program is executed in a machine with CPU, memory, files and I/O devices**.

POS(0230A)-1.9

Multiple copies of the same program?

There is something missing, though.

- One can run two copies of the same program at the same time, e.g., on different terminals (or "windows").
But having two terminals is not a pre-requisite. It is possible to run two programs on the same terminal if one of them "runs in the background".
- and... the two copies run **independently!** One copy will not "eat" up the input of the other, and the execution of an instruction in one copy has **no effect** on the other. We usually call each such copy a "**process**".
- Processes seem to have independent CPUs, memory, devices, etc.
- But... **we only have one computer**, not many! One computer has only one CPU, a big trunk of memory, and a single set of devices. How all these works?
It seems that there are many machines!

POS(0230A)-1.10

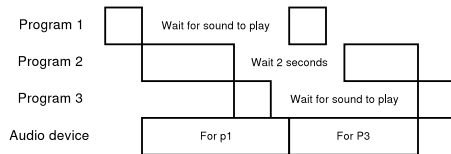
Independent CPU

- First, let's look at part of the problem. How a single CPU can execute multiple processes at the same time?
- Indeed, it is **impossible**. But then it is possible to make an **illusion** that it **seems** to happen.
- The idea: **switch among all the processes** quickly enough so that you don't notice that the CPU is not fully used to run the program!
How quick is quick is enough? Typically, it is around 10–1000 times a second.
- So **one** CPU can work on many processes. But... who decide which to run at each time?
- It's the **scheduling** problem. Let's delay thinking about it for a moment.
We still have other bigger problems to solve...

POS(0230A)-1.11

Overlapped CPU and I/O

- Let's go a bit further to understand **why we want to run multiple programs** to be running concurrently in the same computer.
- Answer 1: they **keep the CPU busy** rather than waiting for I/O.



The CPU work is done when the I/O for another program is proceeding. We say that the CPU and I/O work **overlap**. E.g., first part of CPU work of program 2 overlap with the audio device work for program 1.

- Answer 2: they can then **serve multiple users**.

POS(0230A)-1.12

Independent memory and context

- How the program can have independent memory and execution?
- Memory:** When the CPU switches to a process, it must see its own variables, i.e., its own memory.
- Context:** When the CPU switches to a process, it must continue to run the code *at the point where it left off* the last time when the CPU switches from a program.
The context is stored in an "instruction pointer" and a "stack", as you should have learnt in the MOA course.
- To have independent memory, each process use different portion of the big trunk of physical memory.
- To have independent context, each process owns a small piece of memory, which temporarily stores the registers when the CPU is not running the program.

POS(0230A)-1.13

How all these are implemented?

The programs do more things than a programmer normally expects:

- They give up the CPU from time to time for other programs to run.
- Before giving up the CPU, they store their context. Upon acquiring the CPU, they restore their context.
- If the program access memory of another program due to bugs, they try to catch the problem without affecting the other program.

But we write none of these in our programs! Two possibilities:

- The **compiler** does all these. (But what if we write machine code?)
- The reality: Our programs are not running **directly** in the machine.

POS(0230A)-1.14

Virtualization

- Only one program, the "**OS Kernel**", runs **directly** on the computer. It controls all processes.
- Other programs do **not** run (directly) in the "real" machine. Instead, they run in an environment **prepared** by the OS kernel.
We call such an environment a "process", overloading the word a bit.
- The OS kernel will **configure the computer** (CPU, memory, devices, etc) for each program to run, but **retain control**.
It **forces** all programs to do the things we mentioned in the last slide.
- If a process needs to access a device, etc., it **must** ask the OS kernel to do it—through the **system call interface**.
In general, anything that might affect other processes is done by system calls.
- A process is a **virtual machine** on which our programs run on.
The book use the word "virtual machine" a bit differently, but we will consistently stick to "process=virtual machine" throughout this course.

POS(0230A)-1.15

Why it is a good idea?

- The OS kernel is the central body that deals with resources allocation. Various **optimizations** can be done to make it **efficient**.
- User programs need not pay special care about the existence of other processes, so it is more **convenient**.
- All processes are guaranteed to be **separated**, so it is easy to trace problems to their causes.
- Complicated device manipulations can be **hidden** by the OS kernel, making it **easier to program**.
The OS allows (forces) us to access files rather than actual devices like disks.
- Policies about "who can perform what" can be **enforced** by the OS kernel, rather than by carefully auditing all programs.

POS(0230A)-1.16

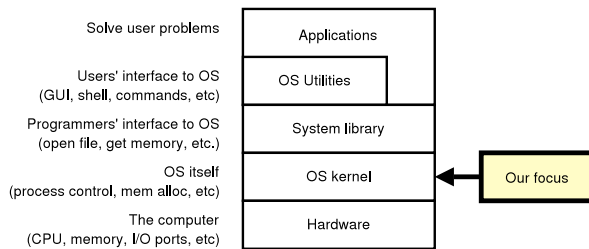
Overview of services provided by an OS

- Process management:** represent and keep track of virtual machines.
- Process communication:** let virtual machines talk with each other.
- CPU scheduling:** select a virtual machine to run.
- Memory management:** give memory to virtual machines.
- Filesystem:** let virtual machines save results.
- I/O device handling:** let virtual machines do I/O.
- Network management:** let virtual machines share the network.
- Protection:** decide which virtual machine can access what resource.

POS(0230A)-1.17

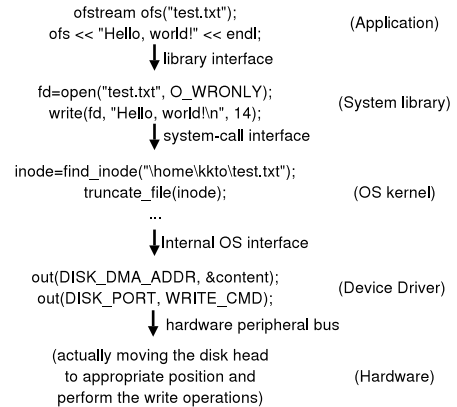
Typical structure of OS

The Operating System (OS) refers to the OS kernel together with its supporting software, usually including the system library and utilities.



POS(0230A)-1.18

Example



POS(0230A)-1.19

A quick historical overview of OS

- **Early systems:** acts as program loader and device driver.
- **Simple Batch System:** Load a program when executing another.
- **Multiprogrammed Batched Systems:** Load many programs to memory at the same time, run a program when another is waiting for I/O.
This is a big change! Most of the OS concepts arises directly from running multiple programs at the same time.
- **Time-Sharing Systems:** Interrupt jobs involuntarily to create an illusion of concurrency. Implemented user separation and multi-processing.
- **Desktop Computer:** Reduce the complexity by removing user separation and multi-processing.
But everything removed are eventually pulled back in again.

POS(0230A)-1.20

New trends

OS designs must change to meet new challenges.

- **Parallel Systems:** Use many CPUs to build a fast computer.
- **Clusters:** Use many computers to build a cheap but fast system.
- **Distributed Systems:** Make a network of computers look like one big computer.
- **Real time Systems:** Provide timing guarantees.
- **Hand-held Systems:** Reduce a computer to pocket size.

These are advanced topics, and our course will mainly focus on time-shared multi-user systems.

POS(0230A)-1.21

Some popular OS

- Early research work: **Multics**. Never really there, but all the great ideas of multitasking, multiprocessing and multiuser come here.
It is so complex that development never finishes before it is abandoned.
- Unix workstation and server: **Solaris, FreeBSD, MacOSX**. Stripped Multics so that it can be reasonably implemented.
- Unix-like OS: **Linux, Mach/GNU Hurd**. Make Unix free.
- Desktop "OS": **DOS, Windows 98/ME**. "Library" is probably a better word than "OS", as protection is real weak.
- Microsoft server OS: **Windows NT/2000/XP**. A real OS that looks a bit like DOS and Windows.
- Distributed OS: **Amoeba, Sprite**. Designed primarily for research.
- PDA (hand-held) "OS": **PalmOS, Windows CE**. Again, sort of "library".
The power-saving CPU does not provide most protection mechanisms.

POS(0230A)-1.22

Lecture schedule

Preparation:

- Wk 1: Introduction
- Wk 2–3: External interface

Core Mechanisms :

- Wk 4: User and kernel mode
- Wk 5: Process Internal
- Wk 6: Filesystem interface

Independent Subsystems / OS Theory:

- Wk 7: Virtual Memory
- Wk 8: Program and library loading, Quiz
- Wk 9: Synchronization
- Wk 10: Deadlocks
- Wk 11: CPU Scheduling
- Wk 12: User separation
- Wk 13: I/O and Filesystem implementation

POS(0230A)-1.23

Chapter dependencies

